Assignment 7 (Final Project):

Bulletin Board (Part 2)

Collaboration Policy

You may not use code from any source (another student, a book, online, etc.) within your solution to this assignment. In fact, you may not even look at another student's solution or partial solution to this assignment. You also may not allow another student to look at any part of your solution to this exercise. You should get help on this assignment by coming to the instructor's or TA's office hours or by posting questions on Piazza (you still must not post assignment code publically on Piazza.) See the full Course Collaboration Policy here: Collaboration Policy

---

This final project will be worth the same points as 2 programming assignments (200 points).

Problem Definition

In this assignment, you will be extending your bulletin board project, so first make sure you have your previous BBoard assignment working perfectly (you can user tinker assignment to help you fix it).

We will be adding two major new functionalities:

1. The bulletin board will now be able to handle replies to replies, and will display them indented according to their "level" (e.g. Replies are indented 2 spaces; Replies to Replies 4 spaces; Replies to Replies to Replies 6 spaces; etc.) You will use recursive techniques to display this.
2. The board will now be able to save the entire board (all Message content) to file, and read that file back in; as well as reading in User data from file as you did with the previous bulletin board assignment.

You may re-use your code from the previous BBoard assignment (with modifications, of course) - but as always using another student's code will trigger disciplinary action.

File Naming Specifications

The files that you submit should be named:
main.cpp, **BBoard.h**, **BBoard.cpp**, **Message.h**, **Message.cpp**,**Topic.h**, **Topic.cpp**, **Reply.h**, **Reply.cpp**, **User.h**, **User.cpp** , and **Makefile**.

The names are case-sensitive.

Class Specifications

BBoard Class interface

You may not change the public interface or the encapsulated data in any way; you may modify or omit any of the suggested private "helper" functions, or add new ones.

Private Data Fields

- title - string : *title of the board; initialized via constructor*
- user_list - vector<User> : *list of members; initially empty, then populated via load_users()* ***(Note name change from previous assignment!)***
- current_user (***Major change from previous assignment!***) - const User * : *this is now a **pointer** to the user who is currently logged in; it is initialized by the constructors to **0 (NULL)**, then set via login()*

  REMEMBER to change all your current_user.get_username() to current_user->get_username()
- message_list (***Major change from previous assignment!***) - vector<Message *> : *list of message pointers, initially empty*

Public Member Functions

This is the public interface of the class - the contract with the user.

BBoard( )
default constructor that creates a board with a default title, empty user & message lists, and a no current_user (what should pointer be set to if there is nothing to point to?)

BBoard(const string &ttl)
same as default c'tor, except that it sets the title of the board.

~BBoard( )

destructor that deallocates all Messages pointed to by message_list
(THINK CAREFULLY ABOUT THIS!!)

bool load_users(const string &userfile) ***Name Change !!!***

This replaces the "setup" function in the previous assignment.

This function gets a filename (userfile) of a file that stores the user info in the given format (See Input File Format Specs).

It opens and reads the file of all authorized users and passwords, constructs a User object from each name/password pair, and populates the user_list vector.

If an error occurs when opening the file, it returns false. Returns true otherwise.

bool load_messages(const string& datafile) New !!!

This function gets a filename (datafile) of a file that stores the messages from previous sessions in the given format (See File Format Specs).

It opens and reads the file, creating Topic and Reply objects as appropriate, and fills the message_list vector (note: remember that message_list is a vector of pointers, not messages).

If an error occurs when opening the file, it returns false. Returns true otherwise.

See below (Tips and Tricks) for tips.

bool save_messages(const string& datafile) New !!!

This is the final action of the bulletin board before closing:

It gets a filename (datafile) of a file that will store all of the the messages, and after opening the file writes the messages into it with the same format (See File Format Specs).

If an error occurs when opening the file, it returns false. Returns true otherwise.

(Note: Since you will be opening the file for writing - i.e. an ofstream - "file not found" is NOT an error; a new file will simply be created for you).

See below (Tips and Tricks) for tips.

void login( ) (*Major changes from previous assignment!*)

asks for and validates current user/password

This function asks for a username and password, and checks the user_list vector for a matching User.

If a match is found, it sets current_user to the identified User from the list (remember,

*current_user is now a **pointer**, not an actual object).*

If not found, it will keep asking until a match is found or the user types: 'q' or 'Q' as the username (you may assume we do not have a member with name 'q' or 'Q')

When the user chooses to quit, say "Bye!" and just return from the login function - meaning that in main(), current_user will be unchanged from its initial value, which should have been set to 0.

See the output specifications for details.

void run( ) *(No direct change, but see the private display() and add_reply() helper functions)*

This function includes the main loop of the BBoard.
***If and only if** there is a valid current_user, enter the main loop, displaying the menu items:*

- Display Messages ('D' or 'd')
- Add New Topic('N' or 'n')
- Add Reply ('R' or 'r')
- Quit ('Q' or 'q')

The user should select one of these menu items, which should then invoke the corresponding method of BBoard; with any other input, the user should be asked to try again.

'Q'/'q' should cause the run function to end (return).

Note: if login() did not set a valid current_user, this function must immediately exit without showing the menu - i.e. there must be no way to get around logging in!

See suggested private functions for more details.

***Suggested* Private Member Functions**

These are "helper functions": member functions that will never be needed by a user of the class - they are needed by the public interface functions. You are free to change/add/remove these functions as you wish:

void add_user(const string& name, const string& pass) (*No change from previous assignment*)

This function may be called from the load_users function to add the users from file to user_list.

~~**bool user_exists(const string& name, const string& pass) const**~~ **(*No longer needed*)**

const User * get_user(const string &name, const string &pw) const

This function includes the functionality of the old "user_exists" helper:

It traverses the user_list, testing for the existence of a user with the given name and password; if this user is NOT found, the function returns 0 (address 0); otherwise it returns a ***pointer*** to the identified User

(the statement return &user_list.at(i) *will work - make sure you understand why!!)*

This function may be used by login() to set the current_user (which is now, obviously, a pointer, not an object).

There must be ***no other*** way to set current_user to anything other than 0 (address 0).

void display() const Major Change !!!

This function will traverse the BBoard's message list, and invoke the print function on Topic objects **ONLY**.

It will then be the responsibility of the Topic object to invoke the print function recursively on its own replies.

The BBoard display function will ignore all Reply objects in its message list.

Question: how will BBoard know which Messages are Topics; and how will it know what argument to pass into the print function?

~~**void add_message()**~~ **(*No longer needed*)**

void add_topic() New !!!

This function asks the user to create a new Topic (i.e. the first message of a new discussion "thread"). Every Topic includes a subject (single line), and a body that may consist of ***multiple*** lines. e.g.,

```
Subject: "Thanks"
Body: "I would like to thank you for this awesome board.
       I'll visit here regularly."
```

The body ends when the user enters an empty line (i.e. a "double enter"). Each Topic also stores the username of current_user; and a message id, which is
(index of its Message* **+ 1**)

For example, the first message on the board will be a Topic whose pointer will be stored at index 0 of the message_list vector, so its message id will be (0 + 1) = 1

(there are better ways of establishing unique ids for a set of objects, but for now this will work fine)

Once the Topic has been constructed, a pointer to it is added to message_list.
Hint: Do you need pointers to automatic or dynamic Topics?

void add_reply() New !!!

This function asks the user to enter a reply to a given Message (which may be either a Topic or a Reply, so we can handle nested replies).

The add_reply function first asks the user for the *id of the Message to which they are replying*; if the number provided is greater than the size of message_list it should output an error message and loop back, continuing to ask for a valid Message id number until the user enters either -1 (or any negative number, to return to the menu); or a valid id.

If the id is valid, then the function asks for the body of the new message, and constructs the Reply, pushing back a pointer to it on the message_list

The subject of the Reply is a copy of the parent Topic's subject with the "Re: " prefix.

e.g., suppose the subject of message #9 was "Thanks", and a user is replying to that message:

```
Enter Message ID: 9
Body: It was a pleasure implementing this.
      I hope everyone likes it.
```

Note: *As before, the body ends when the user enters an empty line.*

The above dialog will generate a reply that has "Re: Thanks" as its subject and "It was a pleasure implementing this.\nI hope everyone likes it." as its body.

How will we know what Topic this is a reply to?

In addition to keeping a pointer to every Message (whether Topic or Reply) in BBoard's message_list vector, every Message (whether Topic or Reply) will also store a vector of pointers to all its own Replies.

So whenever we build a Reply, we must immediately store a pointer to it inside its ***parent Message***. Therefore, you will:

1. create a dynamic Reply with the input data. The Reply's constructor should set the Reply's subject to "Re: " + its parent's subject.

2. call the add_child function ***on the parent message*** to push_back the Message* (to the new Reply) to the ***parent's*** child_list vector;

3. Finally, push_back the same pointer to ***BBoard's*** message_list.

Note: When the user chooses to return to the menu, **do not call run() again - just return from this add_reply function.**

User Class

The User class is unchanged from the previous assignment - just make sure your implementation works correctly. As always, you may not alter the class' public interface!

Message, Topic and Reply Classes

We are going to promote our original Message class to an *abstract base class (it has two pure virtual functions, and thus can never be instantiated)*, and derive two new classes from it: Topic and Reply.

The Topic class works like the previous Message, holding the first posting of a thread, with Reply objects continuing the discussion.

The Reply class will be able to manage *nested* replies, i.e. "Replies to Replies" (to any level), using the technique of *recursion*.

ID: The "id" of all Topic and Reply objects is just their "number" - i.e. their (index in message_list + 1). This means that the ids of Replies may not be contiguous with their parent:
e.g. two Replies to say message id #9 might be numbered #34 and #61

Implement the Message, Topic and Reply classes according to the following interfaces:
As always, you may not alter any PUBLIC member functions, or PRIVATE/PROTECTED member variables

Message.h

```
  //inclusion guards
  //includes

  class Message  // abstract base class
  {
   // protected will allow access to these members by objects of derived classes
   protected:
     string author;
     string subject;
     string body;
     unsigned id;  // New !!
     // This will be the size of the message_list vector to which the
     // newly constructed Message * is being pushed_back
     vector<Message *> child_list; // New !!
     // This is how a Message is able to keep track of its Replies

   public:

     // default constructor
     Message();

     // Parameterized constructor;
```

```
    // id will be the size of current BBoard's message_list
    Message(const string &athr,
            const string &sbjct,
            const string &body,
            unsigned id);

    // Be very careful here: some Messages will have two pointers to
    // them, stored in very different places!
    // Where are they, and who should be responsible for deleting
    // them?
    virtual ~Message();

    // returns true if the object is a Reply.
    virtual bool is_reply() const = 0;

    virtual string to_formatted_string() const = 0; // New!!

    /* This function is responsible for printing the Message
     * (whether Topic or Reply), and and all of the Message's
     * "subtree" recursively:
     * After printing the Message with indentation n and appropriate
     * format (see output details), it will invoke itself
     * recursively on all of the Replies in its child_list,
     * incrementing the indentation value at each new level.
     *
     * Note: Each indentation increment represents 2 spaces. e.g. if
     * indentation == 1, the reply should be indented 2 spaces, if
     * it's 2, indent by 4 spaces, etc.
     */
    void print(unsigned indentation) const; // New !!

    //returns the subject string.
    const string & get_subject() const;

    // returns the id.
    unsigned get_id() const; // New !!

    // Adds a pointer to the child to the parent's child_list.
    void add_child(Message *child); // New !!

};

//end inc guards
```

Topic.h

```
//inclusion guards
//includes

class Topic : public Message
{
 // no new member variables
 public:

   //default constructor
```

```
    Topic();

    //Parameterized constructor
    Topic(const string &athr,
          const string &sbjct,
          const string &body,
          unsigned id);

     virtual bool is_reply() const;  // Returns false

    /* to_formatted_string writes the contents of the Topic to a
     * string in the following format:

       <begin_topic>
       :id: 4
       :subject: single line
       :from: author
       :children: 5 6 [this line should not be printed if there are no children.]
       :body: multiple lines - line 1
       line 2
       line 3
       <end>

     * line starting with :children: has the id's of all children (See file specs.
     * for details)
     * This function should not assume the last character in body is a newline.
     * In other words, this function must manually add a newline after the last
     * line of the body (line 3 in this example).
     * Also, the last character in the string must be a newline.
     */
    virtual string to_formatted_string() const;  // New !!

};

//end inc guards
```

Reply.h

```
//inclusion guards
//includes

class Reply : public Message
{
 // no new member variables

 public:
   //default constructor
   Reply();

   /* Parameterized constructor - similar to Message's constructor except:
    * The subject should be initialized to "Re: <sbjct>" not <sbjct>.
    */
   Reply(const string &athr,
         const string &sbjct,
         const string &body,
```

```
            unsigned id);

    virtual bool is_reply() const; // Returns true

    /* to_formatted_string writes the contents of the Reply to a string in the
     * following format:

      <begin_reply>
      :id: 4
      :subject: single line
      :from: author
      :children: 5 6 [this line should not be printed if there are no children.]
      :body: multiple lines
      2nd line
      <end>

     * the line starting with :children: has the list of id's of all children
     * (See file specs. for details)
     * This function should not assume the last character in body is a newline.
     * In other words, this function must manually add a newline after the last
     * line of the body (line 3 in this example).
     * Also, the last character in the string must be a newline.
     */
    virtual string to_formatted_string() const; // New !!

  };

  //end inc guards
```

Main Function

Use the main function, below, to run your work with.

main.cpp

```
//includes

int main(int argc, char **argv)
{
    // check commandline arguments
    // (see zyBook Chapter 8, section 8.7 for explanations and examples)
    if (argc != 3){
        cout << "ERROR: Invalid program call." << endl
            << "Usage: <program_name> userfile datafile" << endl;
        return 1;
    }
    string userfile(argv[1]);
    string datafile(argv[2]);

    BBoard bb("CS12 Bulletin Board");
```

```
    // load users from file
    if (!bb.load_users(userfile))
    {
        cout << "ERROR: Cannot load users from " << userfile << endl;
        return 1;
    }

    // load messages
    if (!bb.load_messages(datafile))
    {
        cout << "ERROR: Cannot load messages from " << datafile << endl;
        return 1;
    }

    bb.login();
    bb.run();

    // save messages
    if (!bb.save_messages(datafile))
    {
        cout << "ERROR: Cannot save messages to " << datafile << endl;
        return 1;
    }

    return 0;
}
```

Input Specifications

Keyboard Input Specifications ***(Same as before)***

- When the user is asked to select a menu item, the only valid inputs are 'D', 'd', 'N', 'n', 'R', 'r, 'Q' and 'q' - not "quit","new", "Quit" etc.
- See sample output for details.

File Format Specifications:

User File Specifications:

The user file is formatted in the following way:

```
user1 pass1
user2 pass2
user3 pass3
...
end
```

The number of users is not fixed.

Use ifstream to read from file. An example file is below:

users1.txt

```
ali87   8422
ricq7   bjk1903
messi   buneyinnessi
mike   ini99ou
jenny   Y00L11A09
end
```

Message File Specifications:

The format of the file should be as below:

```
numberofmessages
<begin_topic>
:id: 1
:subject: single line
:from: author
:body: multiple lines
line 2
<end>
<begin_topic>
:id: 2
:subject: single line
:from: author
:children: 3 4 //This line is omitted if there are no child
               //messages. See first and last blocks.
:body: line 1
line 2
line 3
<end>
...
...
<begin_reply>
:id: 4
:subject: single line
:from: author
:body: line1
<end>
```

Here is an example message file you can use on your tests.

data1.txt

```
10
```

<begin_topic>
:id: 1
:subject: CS12 Assignment 7
:from: messi
:children: 6 9
:body: The assignment is hard so go step by step.
You can read the Tips & Tricks part for some help.
Has anyone started already?
<end>
<begin_topic>
:id: 2
:subject: Your favorite TV show
:from: ricq7
:children: 3 5
:body: So guys, what is your favorite tv show?
<end>
<begin_reply>
:id: 3
:subject: Re: Your favorite TV show
:from: mike
:children: 4
:body: Game of Thrones is an awesome show but there are too
many characters to remember.
We even did not see half of the families and still at least 20
important characters are introduced already.
Is there anyplace that I can read about the characters?
<end>
<begin_reply>
:id: 4
:subject: Re: Re: Your favorite TV show
:from: ricq7
:children: 8
:body: Well the book is one hell of a resource.
And, the script is written very loyal to the text.
<end>
<begin_reply>
:id: 5
:subject: Re: Your favorite TV show
:from: ali87
:children: 7
:body: Whose line is it anyway!!!
Too bad, it is no more.
<end>
<begin_reply>
:id: 6
:subject: Re: CS12 Assignment 7
:from: messi
:body: BUMP

```
<end>
<begin_reply>
:id: 7
:subject: Re: Re: Your favorite TV show
:from: jenny
:body: The cat!
Colin Mochrie is hilarious.
<end>
<begin_reply>
:id: 8
:subject: Re: Re: Re: Your favorite TV show
:from: mike
:body: I'll check that.
Thanks.
<end>
<begin_reply>
:id: 9
:subject: Re: CS12 Assignment 7
:from: messi
:body: BUMP
<end>
<begin_topic>
:id: 10
:subject: Towel Day
:from: messi
:body: Bring your towels on May 25.
See here: http://www.towelday.org/
<end>
```

Using the files

How to Run the Program with users1.txt and data1.txt

We are again using command line arguments: See zyBook Chapter 8, section 8.7 for details

```
./a.out users1.txt data1.txt
```

Output Specifications

Read the examples below and make sure the output matches exactly with the sample runs below (with the same inputs) - in particular, note the differences between Topics and Replies.

Sample run 1

```
$./a.out users1.txt data1.txt

Welcome to Jack's Amazing Bulletin Board
Enter your username ('Q' or 'q' to quit): muzo
Enter your password: cs12
Invalid Username or Password!

Enter your username('Q' or 'q' to quit): ali87
Enter your password: 8422

Welcome back ali87!

Menu
 - Display Messages ('D' or 'd')
 - Add New Topic ('N' or 'n')
 - Add Reply to a Topic ('R' or 'r')
 - Quit ('Q' or 'q')
Choose an action: D

--------------------------------------------------------------------------------
Message #1: CS12 Assignment 7
from messi: The assignment is hard so go step by step.
You can read the Tips & Tricks part for some help.
Has anyone started already?

  Message #6: Re: CS12 Assignment 7
  from messi: BUMP

  Message #9: Re: CS12 Assignment 7
  from messi: BUMP
--------------------------------------------------------------------------------
Message #2: Your favorite TV show
from ricq7: So guys, what is your favorite tv show?

  Message #3: Re: Your favorite TV show
  from mike: Game of Thrones is an awesome show but there are too
  many characters to remember.
  We even did not see half of the families and still at least 20
  important characters are introduced already.
  Is there anyplace that I can read about the characters?

    Message #4: Re: Re: Your favorite TV show
    from ricq7: Well the book is one hell of a resource.
    And, the script is written very loyal to the text.

      Message #8: Re: Re: Re: Your favorite TV show
      from mike: I'll check that.

  Message #5: Re: Your favorite TV show
  from ali87: Whose line is it anyway!!!
  Too bad, it is no more.
```

```
    Message #7: Re: Re: Your favorite TV show
    from jenny: The cat!
    Colin Mochrie is hilarious.
--------------------------------------------------------------------------------
Message #10: Count down to Towel Day!!
from messi: Bring your towels on May 25.
See here: http://www.towelday.org/
--------------------------------------------------------------------------------

Menu
 - Display Messages ('D' or 'd')
 - Add New Topic ('N' or 'n')
 - Add Reply to a Topic ('R' or 'r')
 - Quit ('Q' or 'q')
Choose an action: R

Enter Message ID (-1 for Menu): 5
Enter Body: I heard ABC is thinking of reviving the series.
They should definitely hire everyone back.
Menu
 - Display Messages ('D' or 'd')
 - Add New Topic ('N' or 'n')
 - Add Reply to a Topic ('R' or 'r')
 - Quit ('Q' or 'q')
Choose an action: R

Enter Message ID (-1 for Menu): 10
Enter Body: I did not know about that. It looks fun.
I'll order this one immediately: http://www.royal-plus.de/dna/indexE.html !!
Menu
 - Display Messages ('D' or 'd')
 - Add New Topic ('N' or 'n')
 - Add Reply to a Topic ('R' or 'r')
 - Quit ('Q' or 'q')
Choose an action: R

Enter Message ID (-1 for Menu): 0
Invalid Message ID!!

Enter Message ID (-1 for Menu): 15
Invalid Message ID!!

Enter Message ID (-1 for Menu): -1

Menu
 - Display Messages ('D' or 'd')
 - Add New Topic ('N' or 'n')
 - Add Reply to a Topic ('R' or 'r')
 - Quit ('Q' or 'q')
Choose an action: D

--------------------------------------------------------------------------------
Message #1: CS12 Assignment 7
```

```
from messi: The assignment is hard so go step by step.
You can read the Tips & Tricks part for some help.
Has anyone started already?

  Message #6: Re: CS12 Assignment 7
  from messi: BUMP

  Message #9: Re: CS12 Assignment 7
  from messi: BUMP
--------------------------------------------------------------------------------
Message #2: Your favorite TV show
from ricq7: So guys, what is your favorite tv show?

  Message #3: Re: Your favorite TV show
  from mike: Game of Thrones is an awesome show but there are too
  many characters to remember.
  We even did not see half of the families and still at least 20
  important characters are introduced already.
  Is there anyplace that I can read about the characters?

    Message #4: Re: Re: Your favorite TV show
    from ricq7: Well the book is one hell of a resource.
    And, the script is written very loyal to the text.

      Message #8: Re: Re: Re: Your favorite TV show
      from mike: I'll check that.

  Message #5: Re: Your favorite TV show
  from ali87: Whose line is it anyway!!!
  Too bad, it is no more.

    Message #7: Re: Re: Your favorite TV show
    from jenny: The cat!
    Colin Mochrie is hilarious.

    Message #11: Re: Re: Your favorite TV show
    from ali87: I heard that ABC is thinking on reviving the series.
    They should definitely hire everyone back.
--------------------------------------------------------------------------------
Message #10: Count down to Towel Day!!
from messi: Bring your towels on May 25.
See here: http://www.towelday.org/

  Message #12: Re: Count down to Towel Day!!
  from ali87: I did not know about that. It looks fun.
  I'll order this one immediately: http://www.royal-plus.de/dna/indexE.html !!
--------------------------------------------------------------------------------

Menu
 - Display Messages ('D' or 'd')
 - Add New Topic ('N' or 'n')
 - Add Reply to a Topic ('R' or 'r')
 - Quit ('Q' or 'q')
Choose an action: Q
```

```
Bye!
```

When the program terminates, data1.txt should have been updated.

So the next time you run the program and login, you'll see all the messages that were added in the previous session.

Compiling and Testing

In this assignment, you are required to write and submit your own Makefile.
Make sure you include all the necessary compile flags: `-W -Wall -Werror -pedantic -ansi`

This is what we will be looking for when grading your work:

- User Class
- Message Class
- Topic Class
- Reply Class
- BBoard::load_users function reads the user info to user_list successfully.
- BBoard::login function works as described and sets the current user.
- BBoard::run function displays the menu items and follows the user's commands successfully as shown in sample runs.
- **Output Requirements:** The output should match **EXACTLY with the samples.**

What to Submit

For this assignment you will turn in the following files (named exactly as shown):

- main.cpp (Your test harness.)
- Makefile
- BBoard.h (with exact interface given in specs.)
- BBoard.cpp
- User.h (with exact interface given in specs.)
- User.cpp
- Message.h (with exact interface given in specs.)
- Message.cpp
- Topic.h (with exact interface given in specs.)
- Topic.cpp
- Reply.h (with exact interface given in specs.)
- Reply.cpp

Step-by-Step Approach

Copy your work from the previous assignment (make sure it works perfectly!)

- Message File Read:
  - Read from a simple file that has only a single Topic, and add the pointer to it to message_list in load_messages.
  - Now read from another simple file that has only one Topic and one Reply to that topic, and add it to message_list in load_messages, and add a pointer to the Reply to the Topic's child_list.
  - Now read in a file with multiple Topics and Replies (you will need a loop, of course!!)
- Message File Write:
  - Build the to_formatted_string() function in Reply (Don't forget newline after <end>)
  - Build the to_formatted_string() function in Topic
  - In BBoard::save_messages() call to_formatted_string() on each Message in message_list and print the returning strings to screen/terminal for debugging.
  - Finally, print the strings to file instead of terminal
- Nested Replies and Display (Leave this to the last, after you have read about recursion)
  - Modify Message.h, Topic.h and Reply.h files as specified
  - Modify the print function in Message.cpp to call itself recursively on each of its children after printing the topic itself
  - In BBoard, modify display and add_reply functions as specified.
  - Have your program run correctly with nested replies

Hints & Tips:

- **Reading from file:**

Be very careful assigning "children" (Replies)!!! The children Messages will not be in memory when you are reading the parent, due to their order in the file.
One way of doing this would be:
For each Message (whether Topic or Reply), maintain a separate string vector that stores the strings following the ":children:" tag (e.g., "5 6 21").
When you finish populating the message_list (or after you close the file), parse the strings (see hint below), construct the Reply objects, and store their POINTERS.
e.g.

```
message_list[i].add_child(message_list[child-1]);
```

for each element (child) in the corresponding string.
Make sure you add an empty string to any string vector with no :children: tag.

Alternatively, you could construct a Message, then traverse that Message's child_list and construct a default (empty) Reply for each entry - just make sure you know where to place their pointers in Bboard's message_list, and that you complete their initialization correctly when you come to their listing in the file.

- **Using stringstream to parse message files**

Review the stringstream section in the online text! (Section 8.4)

```
void parse_for_children( vector<int> &child_list, const string &filename )
```

```
{
    ifstream infile(filename.c_str());
    string discard;
    int child_id;
    string child_id_string;

    // discard <begin_?????> and :children:
    infile >> discard >> discard;

    // WARNING: this function is just a HINT - in your actual program you
    // will need to know when you are dealing with a reply or a topic

    getline(infile, child_id_string); // get string containing all child ids

    stringstream iss(child_id_string);
    while (iss >> child_id)
        child_list.push_back(child_id);
}
```